Noncontiguous MPI-IO Performance on PVFS

Rob Latham, Rob Ross

Abstract

Performance could be better.

1 Introduction

On April 10, 2003, a PVFS user reported particularly low performance for a modified noncontiguous testing application that they were using to test their system. This led us into an investigation of the problem and the implementation of solutions.

This resulted in a series of experiments to determine exactly what was going on and to hopefully "fix" the problem.

This highlights the complexities in MPI-IO implementations. Maybe this is useful as part of a paper later on.

2 The `noncontig` Test

By default the test works on noncontiguous regions of 4 bytes (`MPI_INT`s).

2.1 Parameters

- fsize - file size in MB
- elmtcount - number of elements (`MPI_INT`s) in a contiguous chunk
- displs - emulates a "header" on the data, size in bytes
- cbsize - size in KB passed as the "cb_buffer_size" hint
- coll - use collective MPI-IO calls rather than independent ones
- bufsize - size of memory buffer, in MB

3 ROMIO Hints and I/O Methods

So there are actually three ways (at least) that a collective call can be serviced:

- no two-phase, using contiguous I/O
- no two-phase, using noncontiguous I/O (for PVFS, listio)
- two-phase

You can ensure that the third one WON'T happen by setting "romio_cb_write" to "disable", or that it *will* with an "enable".

By default the two-phase optimization is applied if the accesses of the processes are interleaved, but not otherwise.

And it gets trickier! There's also some aggregation control going on. By default ROMIO uses exactly one process per node for I/O in the collective case. See the notes on "cb_config_list" in the ROMIO User's Guide. This is probably not helping your performance either in this particular case.

3.1 Further Study

Further investigation noted a number of things.

First, in the read case data sieving will be used for cases where the data is noncontiguous in file. It is not used in the case where the data is contiguous in file. In all these cases, the macro `ADIOI_BUFFERED_READ` is used, which does a lot of memory allocation and freeing.

In the contig/contig case, data is read straight into the user's buffer.

For writes, data sieving is not used with PVFS. Instead, by default, a "naive" write approach is used for cases where data is noncontiguous in file. For the case where data is contiguous in file but noncontiguous in memory, `pvfs_writev` is used instead. For contig/contig cases the data is read straight into the buffer.

From Joachim's data, and subsequent tests, we see:

- read performance is typically considerably lower for all noncontiguous file cases than the contiguous ones (and later we see that contig in file but noncontig in memory is pretty bad too)
- write performance is bad in all cases but contig/contig, but is particularly bad in the case where `pvfs_writev` is used (contiguous in file, noncontiguous in memory)

From this we determined that we should do a quick parameter sweep to try to isolate some of the artifacts seen here. In particular we want to see how much of an effect data sieving is having on performance, what impact this 4 byte element size is having, and how the more recent "list I/O" code impacts performance. In the process too we quickly realized that the use of `pvfs_writev` was a poor one, and implemented a fix for this.

By default, ROMIO only applies the two-phase optimization in cases where data is "interleaved", that is, the regions of different processes overlap. So we would expect that use of collective I/O would only really be of benefit in cases where data is noncontiguous in file.

4 Testing

Testing was performed with PVFS version 1.5.6-pre3 and ROMIO 1.2.5. Debugging was enabled in the ROMIO code, leading to some small amount of performance degredation.

All tests were run on the data grid nodes. There are 19 I/O nodes on the data grid nodes at MCS. All tests were run with 8 compute processes, on separate nodes which were also serving as I/O nodes.

Figure 1: Reads, noncontiguous in memory and contiguous in file

File size was fixed at 2MB. Veclen times elmtcount was fixed at 1024 (so as we increase elmtcount we decrease veclen, thus maintaining a fixed file size). The elmtcount was tested between 1 and 128.

In all graphs we show the peak performance obtained for accesses that were contiguous in file and memory as a horizontal line at the top of the graph in order to provide perspective on relative performance. [Should we normalize values instead?]

5 Future Work

Get rid of crufty buffered read code in ROMIO!

Appendix A: The Email

```
Date: Thu, 10 Apr 2003 16:46:55 +0200
From: Joachim Worringen <joachim@ccrl-nece.de>
To: pvfs-users@beowulf-underground.org
Subject: [PVFS-users] low performance for non-contiguous MPI-IO


Hi all,

I have written a small benchmark for non-contiguous MPI-IO. For a certain case
of collective writes, I observe *very* low performance on our pvfs file
system (in the range of 10kB/s). Other test cases behave normal on pvfs, too,
and get some MB/s.
```

These are the numbers for pvfs, the arrow marks the suspect number :

```
# testing noncontiguous in memory, noncontiguous in file using collective I/O
# vector count = 512 - access count = 512
 write bandwidth (min/max/acc [MB/s]) :  0.362 /  0.529 /  1.947
 read  bandwidth (min/max/acc [MB/s]) :  0.469 /  0.479 /  1.898

# testing noncontiguous in memory, contiguous in file using collective I/O
# vector count = 512 - access count = 512
 write bandwidth (min/max/acc [MB/s]) :  0.010 /  0.010 /  0.041 <====
 read  bandwidth (min/max/acc [MB/s]) :  2.498 /  2.939 / 10.654

# testing contiguous in memory, noncontiguous in file using collective I/O
# vector count = 128 - access count = 128
 write bandwidth (min/max/acc [MB/s]) :  0.394 /  0.599 /  2.192
 read  bandwidth (min/max/acc [MB/s]) :  0.511 /  0.523 /  2.070

# testing contiguous in memory, contiguous in file using collective I/O
# vector count = 128 - access count = 128
 write bandwidth (min/max/acc [MB/s]) :  1.889 /  2.463 /  9.212
 read  bandwidth (min/max/acc [MB/s]) :  3.290 /  3.573 / 13.573
```

On other file systems (UFS, NFS), this case gives "normal" numbers, which means that accessing a non-contiguous file-view is very slow, using a non-contiguous datatype in memory is slow, and if everything is contiguous, its getting fast. The same test run on a local disk:

```
# testing noncontiguous in memory, noncontiguous in file using collective I/O
# vector count = 512 - access count = 512
 write bandwidth (min/max/acc [MB/s]) :  0.415 /  0.533 /  2.013
 read  bandwidth (min/max/acc [MB/s]) :  0.530 /  0.539 /  2.139

# testing noncontiguous in memory, contiguous in file using collective I/O
# vector count = 512 - access count = 512
 write bandwidth (min/max/acc [MB/s]) :  3.752 /  5.871 / 18.165
 read  bandwidth (min/max/acc [MB/s]) :  6.703 /  6.827 / 27.084

# testing contiguous in memory, noncontiguous in file using collective I/O
# vector count = 128 - access count = 128
 write bandwidth (min/max/acc [MB/s]) :  0.460 /  0.620 /  2.320
 read  bandwidth (min/max/acc [MB/s]) :  0.578 /  0.589 /  2.334

# testing contiguous in memory, contiguous in file using collective I/O
# vector count = 128 - access count = 128
 write bandwidth (min/max/acc [MB/s]) : 13.298 / 54.549 / 117.902
 read  bandwidth (min/max/acc [MB/s]) : 32.442 / 36.041 / 135.333
```

I use mpich-1.2.5 and pvfs-1.5.6. Is anybody willing to run this benchmark on his pvfs to validate my observation? Maybe it has something to do with our setup.

I know that due to the missing ability of pvfs to lock files, individual non-contiguous writes are very slow. But this is not the case here as it is a collective call which needs no locking. Also, the fileview used in the suspect case is contiguous.

Joachim